第二十五节 服务注册与发现

Spring Cloud Commons 抽象

- Spring Cloud Bootstrap 应用上下文
- 服务注册与发现抽象
- 负载均衡抽象
- 服务熔断抽象
- 安全
- 配置抽象（分布式）

Spring Cloud 服务注册与发现

@EnableDiscoveryClient 注解

设计模式

@Enable 模块驱动

核心实现 API - org.springframework.cloud.commons.util.SpringFactoryImportSelector

底层 API 运用

ImportSelector API：org.springframework.context.annotation.DeferredImportSelector

Spring Factories API：org.springframework.core.io.support.SpringFactoriesLoader

配置文件：/META-INF/spring.factories

逻辑：在 /META-INF/spring.factories 寻找注解 org.springframework.cloud.client.discovery.EnableDiscoveryClient 的配置：

设计不足

`@EnableDiscoveryClient` 允许不在 /META-INF/spring.factories 资源中配置 org.springframework.cloud.client.discovery.EnableDiscoveryClient 的实现类，所以 Eureka 以及 Nacos 实现均为配置。不过 `@EnableCircuitBreaker` 却配置了实现类。

服务发现 API - DiscoveryClient

接口定义

```
public interface DiscoveryClient extends
Ordered {

    /**
     * Default order of the discovery client.
     */
    int DEFAULT_ORDER = 0;

    /**
     * A human-readable description of the
implementation, used in HealthIndicator.
     * @return The description.
     */
    String description();

    /**
     * Gets all ServiceInstances associated
with a particular serviceId.
     * @param serviceId The serviceId to query.
```

```
     * @return A List of ServiceInstance.
     */
    List<ServiceInstance> getInstances(String
serviceId);

    /**
     * @return All known service IDs.
     */
    List<String> getServices();

    /**
     * Default implementation for getting order
of discovery clients.
     * @return order
     */
    @Override
    default int getOrder() {
        return DEFAULT_ORDER;
    }

}
```

核心概念

- serviceId：服务 ID，成为服务名称，逻辑名称
- ServiceInstance：服务实例，服务节点（类似于Dubbo Node）

设计缺陷

- 缺少分页查询
  - getServices()
  - getInstances(String)

> DiscoveryClient 一次查询的 Payload 会非常大

实现优化：

- 增加 Service ID 和 ServiceInstance 总数
- 增加分页方法

假设：100,000 服务实例，客户端负载均衡，shard 100

一个客户端在 1000 个实例负载均衡

服务实例 API - ServiceInstance

接口定义

```
public interface ServiceInstance {

    /**
     * @return The unique instance ID as
registered.
     */
    default String getInstanceId() {
        return null;
    }

    /**
```

```
     * @return The service ID as registered.
     */
    String getServiceId();

    /**
     * @return The hostname of the registered
service instance.
     */
    String getHost();

    /**
     * @return The port of the registered
service instance.
     */
    int getPort();

    /**
     * @return Whether the port of the
registered service instance uses HTTPS.
     */
    boolean isSecure();

    /**
     * @return The service URI address.
     */
    URI getUri();

    /**
     * @return The key / value pair metadata
associated with the service instance.
     */
    Map<String, String> getMetadata();
```

```
    /**
     * @return The scheme of the service
instance.
     */
    default String getScheme() {
        return null;
    }

}
```

重要方法

- getMetadata()：扩展元信息，补充当前类不足的信息

DiscoveryClient 组合实现 - CompositeDiscoveryClient

使用场景

当 Spring Cloud 应用需要注册中心迁移时，通常会使用到多个 DiscoveryClient 组合，比如 Eureka 迁移到 Nacos 上面。假设，某公司有 3个应用，过去全部注册在 Eureka 上，现在需要 Eureka 逐步迁移到 Nacos 上。先将 1 个应用注册到 Nacos，老的 3 个应用还是注册在 Eureka，当 Nacos 上的 1 个应用运行稳定，逐步将这 1 个应用从 Eureka 注销，并且让 Spring Cloud 服务消费应用连接 Eureka 和 Nacos 注册中心，因为 CompositeDiscoveryClient#getServices() 方法会合并两种注册中心的服务（ID）集合。

举例：

- 迁移前
  - Eureka : A、B、C
  - Nacos：无
- 迁移中（1）
  - Eureka : A、B、C
  - Nacos：A
- 迁移中（2）
  - Eureka : B、C
  - Nacos：A

以此类推，直到 A、B、C 完全迁移

- 迁移后
  - Eureka：无
  - Nacos：A、B、C

云产品方案

- 阿里云 - 微服务引擎（MSE）

底层 Nacos 作为注册中心，兼容 Eureka、Consul 等

Nacos 适配 Eureka 和 Consul 等提供 REST API。

Eureka REST API：https://github.com/Netflix/eureka/wiki/Eureka-REST-operations

Consul API：https://www.consul.io/api-docs

- Nacos Sync

设计模式

- 组合模式：实现者（1）和被组合的成员（N）实现相同的接口（相同类型），通常利用迭代等手段来实现
  - 实现
    - 存在结果实现：N 个成员中，成员 M（1<= M <= N）计算出来结果，将其结果返回，N-M 个成员不再计算
      - 比如：org.springframework.cloud.client.discovery.composite.CompositeDiscoveryClient#getInstances
    - 合并结果实现：N 个成员计算结果合并
      - 比如：
        - org.springframework.cloud.client.discovery.composite.CompositeDiscoveryClient#getServices
    - 流水线实现：不返回结果，N 个成员逐一执行

充当角色

CompositeDiscoveryClient 是 Spring Cloud 服务发现客户端 Primary Bean，如果存在多个 DiscoveryClient 实现，会将它们组合。

```
@Configuration(proxyBeanMethods = false)
@AutoConfigureBefore(SimpleDiscoveryClientAutoC
onfiguration.class)
public class
CompositeDiscoveryClientAutoConfiguration {

    @Bean
    @Primary
    public CompositeDiscoveryClient
compositeDiscoveryClient(
            List<DiscoveryClient>
discoveryClients) {
        return new
CompositeDiscoveryClient(discoveryClients);
    }

}
```

DiscoveryClient 健康指标

核心 API - DiscoveryClientHealthIndicator

Spring Boot Actuator 中存在一个健康检查 Endpoint，/actuator/health，它聚合多方健康指标，来表示整体应用的健康程度，如果有一个 HealthIndicator 存在不健康，那么整体会表示不健康。

DiscoveryClientHealthIndicator 与 DiscoveryClient 是一对一的关系

核心API - DiscoveryCompositeHealthContributor

一个 Spring Cloud 应用允许多个 DiscoveryClient

关联 API

org.springframework.boot.actuate.health.Health

org.springframework.boot.actuate.health.Status

org.springframework.boot.actuate.health.HealthIndicator

Dubbo 类似 API

org.apache.dubbo.common.status.StatusChecker

服务注册 API - ServiceRegistry

使用场景

Spring Cloud 服务注册 API 通常只能单一注册，可以自定义实现多注册，不过需要关注这个问题

Registration 不是统一 ServiceInstance，比如：

- Eureka 实现 - org.springframework.cloud.netflix.eureka.serviceregistry.EurekaRegistration，
- Nacos 实现 - com.alibaba.cloud.nacos.registry.NacosRegistration

假设实现一个通用的 Registration 实现，那么如何转型成目标的 Registration 实现

通用的 Registration 实现等同于 ServiceInstance 实现，然而特定的 Registration 实现需要增加额外的内容。

CompositeServiceRegistry 组合 EurekaServiceRegistry 和 NacosServiceRegistry，但是两种注册中心通用注册对象，就是Registration 接口实现，无法增加其他额外的属性。

设计缺陷

ServiceRegistry 不应该注册特定注册中心 Registration 接口的实现（不应该泛型），而应该注册 ServiceInstance 抽象实现，然所有注册中心均适配 ServiceInstance 接口实现

接口定义

```
public interface ServiceRegistry<R extends
Registration> {

    /**
     * Registers the registration. A
registration typically has information about an
     * instance, such as its hostname and port.
     * @param registration registration meta
data
     */
    void register(R registration);

    /**
     * Deregisters the registration.
     * @param registration registration meta
data
     */
    void deregister(R registration);

    /**
```

```
     * Closes the ServiceRegistry. This is a
lifecycle method.
     */
    void close();

    /**
     * Sets the status of the registration. The
status values are determined by the
     * individual implementations.
     * @param registration The registration to
update.
     * @param status The status to set.
     * @see
org.springframework.cloud.client.serviceregistr
y.endpoint.ServiceRegistryEndpoint
     */
    void setStatus(R registration, String
status);

    /**
     * Gets the status of a particular
registration.
     * @param registration The registration to
query.
     * @param <T> The type of the status.
     * @return The status of the registration.
     * @see
org.springframework.cloud.client.serviceregistr
y.endpoint.ServiceRegistryEndpoint
     */
    <T> T getStatus(R registration);

}
```

服务注册对象接口 - Registration

接口定义

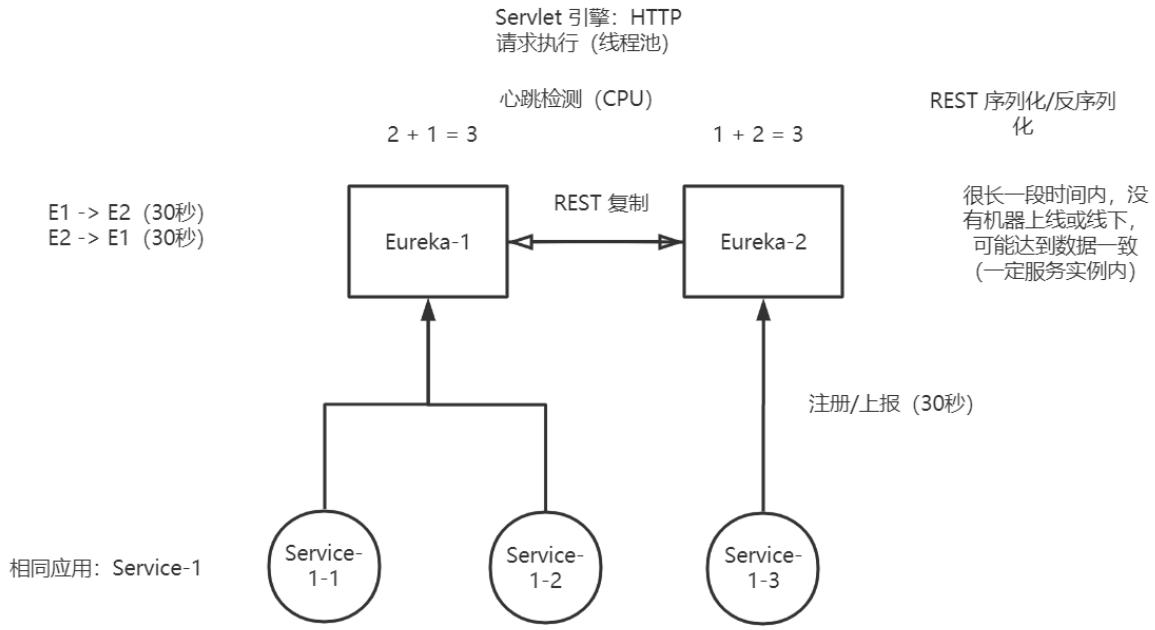
```
public interface Registration extends
ServiceInstance {

}
```

注册中心实现

注册中心	CAP 模型	数据更新	容量范围
Eureka	AP	HTTP 轮训更新、注册（节点复制，最终一致性）	2w - 3w
Zookeeper	CP	ZAB（Leader 更新需要 Follower 确认，才更新）	1w - 2w
Nacos	AP/CP		10w+
Consul	AP/CP		5000

Eureka



问题

1. 关于 Dubbo 泛化调用细节

使用场景

- Dubbo 消费端在没有服务接口的前提下使用
  - 没有 API artifact
    - API ID
      - Service 信息
        - 接口
        - version
        - group
      - 方法名称
      - 方法参数（类型、内容）

- Dubbo 网关
  - Alibaba API 网关（前端 JS 调用后端服务）
    - TOP
      - PC 端
    - MTOP
      - 无线端
- 基于 Dubbo Low Code/ No Code 平台
  - FaaS
    - Spring Cloud Function
  - Serverless

参考如何使用 Dubbo 泛化调用

服务提供方：https://dubbo.apache.org/zh/docs/v2.7/user/examples/generic-service/

服务消费方：https://dubbo.apache.org/zh/docs/v2.7/user/examples/generic-reference/

Dubbo 泛化调用实现

服务端实现代码参考

```
...
// 用org.apache.dubbo.rpc.service.GenericService
可以替代所有接口实现
```

```
GenericService xxxService = new
XxxGenericService();

// 该实例很重量，里面封装了所有与注册中心及服务提供方连
接，请缓存
ServiceConfig<GenericService> service = new
ServiceConfig<GenericService>();
// 弱类型接口名
service.setInterface("com.xxx.XxxService");
service.setVersion("1.0.0");
// 指向一个通用服务实现
service.setRef(xxxService);

// 暴露及注册服务
service.export();
```

实现：基于 Dubbo Filter 来实现

```
@Activate(group = CommonConstants.PROVIDER,
order = -20000)
public class GenericFilter implements Filter,
Filter.Listener {
    ...

}
```

学习资料

小马哥技术周报 - https://space.bilibili.com/327910845/channel/detail?cid=52311